Using conceptual structures in service creation and verification

Erik Reitsma

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands
e.j.reitsma@math.utwente.nl

Abstract:

Requirement specifications for services in intelligent networks are given in rather informal natural language descriptions. These services are implemented as global service logics (GSLs) at the global functional plane. These GSLs consist of service independent building blocks (SIBs). There is a large difference in level of detail and abstraction between the natural language requirement specifications and the implementation in SIBs. Furthermore there is no description language that spans the entire service creation process. It should be possible to translate the initial informal requirement specifications, the final GSLs and all intermediate representations into such a language. The lack of such a description language leads to two problems in the service creation process.

The first problem is in checking the implementation in the GSL against the requirement specification. A method for checking the implementation should use comparable representations of the implementation and of the requirement specification. Unfortunately most formal description methods that are used today are already too abstract, too far away from the natural language requirement specification.

The second problem is in choosing the correct SIBs to be used in the GSL, when only an informal natural language requirement specification is given. These choices are inherent to any programming process. There are no tools available that assist the programmer in composing a GSL from SIBs based on an informal natural language requirement specification.

We propose the use of conceptual structures as a description language for services throughout the service creation process. Conceptual structures have been introduced by Sowa (1984) and are now considered in ANSI standard efforts in the Information Resource Dictionary Systems Committee (ANSI X3H4 IRDS), the Data Interchange and Repositories Committee (ANSI X3T2) and the Formal Description Committee (ANSI X3J21). Conceptual structures emphasize knowledge and understanding, rather than particular implementations. Both natural language and implementation descriptions (such as GSLs) can be mapped into conceptual structures. Well-defined semantics and algorithms for logical deduction are available, because conceptual structures are based on existential graphs.

When the requirement specification, the corresponding GSL and background knowledge are represented in conceptual structures, existing algorithms can be used to check whether the GSL implements the requirement specification. This should help solving the problem of checking the implementation.

The representation in conceptual structures of the requirement specification can be used to assist in choosing appropriate SIBs. This requires that the specifications of the SIBs are represented in conceptual structures also.

In this presentation we give a short introduction into conceptual structures. Then we illustrate how the use of conceptual structures may contribute during the service creation process. Finally we discuss what is necessary to make the power of conceptual structures available in the service creation process.

Using conceptual structures in service creation and verification

Erik Reitsma
University of Twente
The Netherlands
e.j.reitsma@math.utwente.nl

The research was sponsored by and done at:
Intelligent Networks Applications Laboratory,
Ericsson Telecommunicatie B.V., The Netherlands

Service creation today

- Services are written as programs in some language
- Services use Service Independent Building blocks (SIBs) or SIB-like components
  - SIBs are linked
  - SIBs have data added to them
- Requirement specification translated into a program
  - given SIBs
  - new SIBs if necessary and allowed by the platform
- Produced program must be tested against the requirement specification

Service creation today is essentially programming. Sometimes visually oriented languages are used. If stepwise refinement is used, different languages may be used at different levels of refinement.

Functionality may be grouped into SIBs.

- at the level of SIBs as defined in the ITU-T IN CS-*n* recommendations, and
- at the level of INAP Information Flows and Information Elements.

This talk also analyzes the pros and cons for each model and matches the various models to various service creation scenarios, vendor characteristics, and network characteristics, to determine how the models are suited to different service creation and deployment conditions.

The following are some of the issues and criteria are considered within the analysis of the various Service Creation models:

- the characteristics of products that support the various models - ease of use, flexibility, manageability
- the degree to which the models supports vendor independence and control of the service creation process
- the degree to which the SCE supports innovation, i.e. the development of new types of services or service features

Biographic Information:

Robert Willner has been involved with the development of IN systems since 1990. He is responsible for product planning for the international product line within Bellcore's ISCP Intelligent Network Solutions. He joined Bellcore in 1979 after receiving both an MS in Computer Science and a BS in Engineering from UCLA.

Linda Lee has been involved with the development of IN systems since 1991. She is responsible for product planning for the North American product line within Bellcore's ISCP Intelligent Network Solutions. She has been also involved in development of the international IN/AIN product line within Bellcore's ISCP Intelligent Network Solutions. She joined Bellcore in 1989 after receiving an MS in Computer Science from Harvard University.

Solutions tried so far...

Generally: split large problem into smaller parts

- Top-down, stepwise refinement
  - first step from requirements in natural language still difficult
  - bottom of refinement given: SIBs
- Object orientation
  - also first step difficult
  - which classes?

Methodologies from software engineering are used for service creation. These methodologies usually start with a requirement specification that is already quite formal. During the first step, from the requirement specification to this first (semi-)formal requirement specification, implementation choices must sometimes already be made: what functions are grouped, which are the components of the system?

Engineering in general

given:

- *ReqSpecs*
  requirement specification (natural language)
- *ElementSpecs*
  specification of available "building elements", e.g. SIBs (natural or formal language)
- *BackKnowledge*
  background knowledge (usually in natural language (books, manuals) or in mind of the engineer)

required:

- *ImpSpecs*
  specification of implementation, i.e. program and data (usually formal language)
- ***ImpSpecs* + *ElementSpecs* + *BackKnowledge* implies *ReqSpecs***

In engineering, a requirement specification, a description of the building elements and background knowledge are given. The amount of background knowledge may be large. The goal of the engineer is to find a description of the implementation, such that it satisfies the requirement specification, under the assumption of the building element descriptions and the background knowledge.

The specification of the implementation must be at such a level, that an implementation can be built from it. This can be achieved by describing the elements at a level that is low enough. The specification of the implementation will be at that same level.

The requirement specification must be implied by the implementation specification (under the assumption of the building element descriptions and the background knowledge), because then the requirement specification is true if the implementation is realized.

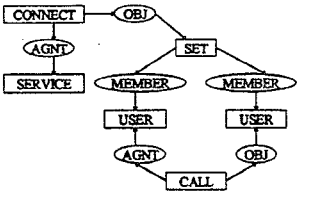
Characteristics of conceptual structures

- Close to natural language
- Emphasis on knowledge, not on modeling
- Logic and deduction based on existential graphs
- Standardization in progress
  - Information Resource Dictionary Systems Committee (ANSI X3H4 IRDS)
  - Data Interchange and Repositories Committee (ANSI X3T2)
  - Formal Description Committee (ANSI X3J21)

Conceptual structures can be used to represent any meaningful statement or statements in natural language. Contrary to most (other) formal languages, the representation is not a model of the (future) system, but a representation of the knowledge about the system. It may describe the elements of the system and the relations between them, but it may also describe characteristics without referring to elements of the system.

Verification requires proof mechanisms over the formalism. In most formal methods knowledge necessary for statements about the system is only implicitly available (e.g. shortest path algorithms, performance calculations, theory on dead locks). Using conceptual structures all such knowledge can be made explicit in the same formalism as used for describing the system itself.

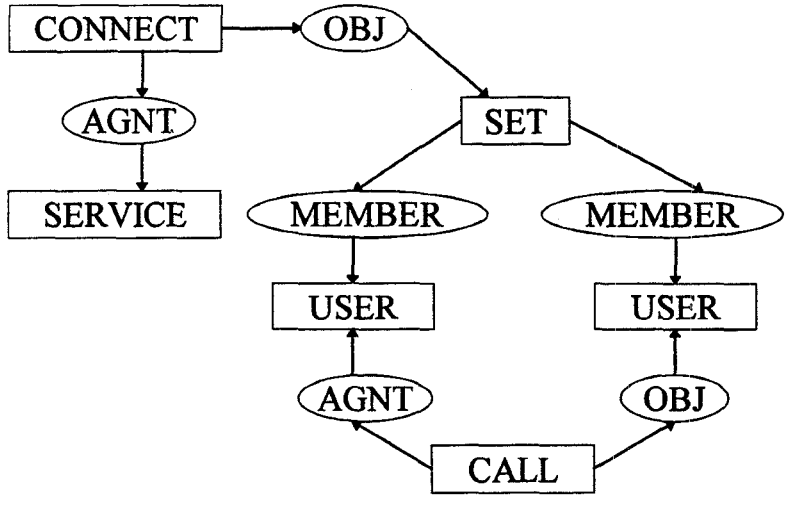
Elements of conceptual structures



- Conceptual graphs
  - concepts
  - relations
- Type hierarchy
- Generic constructs for logical, modal and temporal aspects

Conceptual graphs consist of concepts and relations between them. A concept does not represent a word in the language, but corresponds to a percept in the "real world" (which may be abstract or imaginary...) A relation represents a relationship between concepts.

Example



This conceptual graph expresses the following statement/knowledge:

A service connects a set, which has two members, both users. One of the users calls the other.

This conceptual graph is very simple: there are no logical, modal or temporal aspects.

Concepts and relations have types. Types of concepts may be PHYSICAL_OBJECT, SERVICE, CONNECT, SET, TELEPHONE, SIB, LONG etc., i.e. objects, actions or attributes. Types of relations may be MEMBER, AGENT, CHILD, NOT (unary relation), BETWEEN (ternary relation), etc.

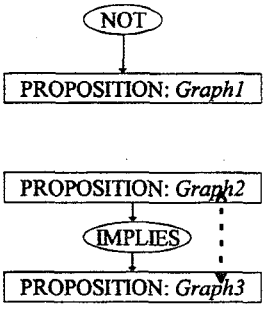
Concepts may have, besides a type, also a marker. An individual marker specifies a specific instance of the type, whereas a generic marker specifies "some" instance of the type.

This means, that a type is somewhat comparable to a class in object orientation, but it allows for generic "instances".

The type hierarchy specifies the subtypes and supertypes of each type. TELEPHONE would be a subtype of TERMINAL and a supertype of ISDN_TELEPHONE.

Logic

- Empty graph is always true
  Conceptual graph alone means: "*this is true*"
- Graphs can be nested into concepts, allowing for meta-statements
- Co-reference links between nested graphs allow identification of concepts



Meta-statements are statements that say something about a proposition. A proposition here should be seen in a very general sense: it could be a statement that is uttered by somebody or the description of a situation. This can be used to express logical statements ("*this* is not true"), modal statements ("*this* may be true") or temporal statements ("*this* happens after *that*"), where *this* and *that* are propositions. Combinations of logical, modal and temporal aspects can be constructed by further nesting of propositions.

Service verification

1. Describe using conceptual structures
   requirements, SIBs, network capabilities, background knowledge
2. Describe implementation using conceptual structures
3. Find proof of requirements
   axioms are SIB descriptions, implementation description, network description and background knowledge
4. Success or failure
   - success: implementation is correct
   - failure: implementation is incorrect or knowledge is missing

Service verification tries to find a proof of the requirements. This can be done using conceptual structures, but this requires that all knowledge is available explicitly in conceptual structures.

Most knowledge can be reused for subsequent problems: for the verification of a new service the description in conceptual structures of the SIBs, the network capabilities and the background knowledge can be reused.

If a proof can not be found, it does not necessarily mean, that the implementation does not satisfy the requirements. Some knowledge may have been overseen, and should be added manually. Each time knowledge is added, the system will produce fewer false failures.

Service creation

1. Describe using conceptual structures
   requirements, SIBs, network capabilities, background knowledge
2. Search for proof of requirements
   axioms are SIBs descriptions, network description and background knowledge and possible implementation choices
3. Success or failure
   - success: used SIBs and their relation in the proof gives implementation
   - failure: missing functionality or other knowledge

Erik Reitsma/University of Twente

Service creation tries to find an implementation that satisfies the requirements. This is more complex than verification. Using conceptual structures, the same proof mechanisms may be used for service creation and service verification. Note that in service creation the requirements must be proven using other axioms than in service verification. In service creation we do not yet know the implementation, but we have several implementation choices. Implementation choices may be using a certain building component in a certain way. When a proof is found, the used possible implementation choices together determine the implementation choices that should be made.

Evaluation/conclusions

Positive

- close to natural language
- support on any level of design
- design can be semi-automated
- knowledge can be reused for other purposes
- theory is available

Negative

- tools are still under development
- amount of necessary background knowledge may be huge
- only theoretical results so far

References

- Sowa, John F., "*Conceptual Structures*", Addison-Wesley Publishing Company, Inc., 1984